Programarea calculatoarelor I (Limbajul C)

G. GRIGORAS

www.infoiasi.ro/~grigoras/Prog1/prog1.html

www.infoiasi.ro/~pc/

Curs 1- Cuprins

- Bibliografie
- Programare – mic glosar
- Limbaj natural vs. limbaj formal
- Programarea ca proces
  - Rezolvarea problemei
  - Implementarea (scrierea programului)
- Exemplu
- O trecere in revista a limbajului C
  - Primul program
  - Expresii

Bibliografie I

- **Al Kelley, Ira Pohl** *A Book on C - Programming in C*, Addison Wesley, Reading, 1998
- **2. Herbert Schildt** *C Manual Complet*, Bucuresti, Ed. Teora, 1998
- **3. Liviu Negrescu** *Limbajele C si C++ pentru incepatori*, vol I- III, Microinformatica, Cluj-Napoca sau Libris Agora, Tirgu Mures
  - Volumul I **-** *Limbajul C*
  - Volumul III **-** *Limbajele C si C++ in Aplicatii*
- **5. E. Horowitz, S. Sahni, S. Anderson - Freed** *Fundamentals of Data Structures in C,* Computer Science Press, 1993

Bibliografie II (http://lib.info.uaic.ro/)

- Brian Brown, An Introduction to C Programming
- Brian Brown, C Programming
- Steve Summit, Intermediate C Programming Class Notes
- Marshall Brain, Introduction to C Programming
- Brian W. Kernighan, Dennis M. Ritchie, Limbajul C

Bibliografie III (compilatoare C, C++)

- O lista de adrese pentru compilatoare C si C++ (free)

http://www.thefreecountry.com/compilers/cpp.shtml

- DJGPP Version 2.03
  http://www.delorie.com/djgpp/
- Borland C++, Version 5.5
  http://www.borland.com/products/downloads/download_cbuilder.html
- Cygwin Project (C, C++)
  http://www.cygwin.com/

Glosar

- problema-solutie (problem-solving): formularea prblemei, gasirea solutiei, exprimarea solutiei
- limbaj de nivel inalt (high-level language)
- limbaj de nivel scazut (low-level language, machine language, assembly language)
- limbaj formal (formal language)
- limbaj natural (natural language)
- portabilitate (portability)
- interpretare (interpret)
- compilare (compile)
- cod sursa (source code)
- cod obiect (object code)

Glosar

- cod executabil (executable code)
- algoritm (algorithm)
- eroare in program (bug)
- sintaxa (syntax)
- semantica (semantics)
- parsare (parse)
- eroare de sintaxa (syntax error)
- exceptie (exception) – eroare la executie
- eroare logica (logical error)
- depanare (debugging)

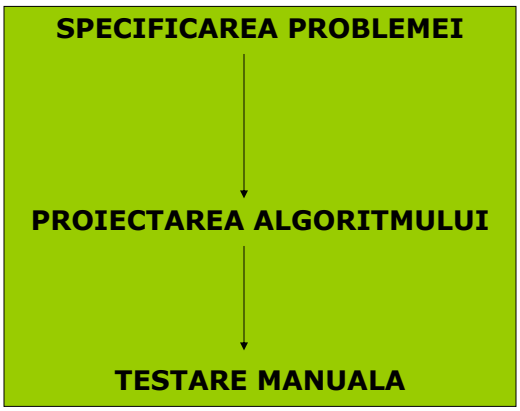
Limbaj natural vs. limbaj formal

- Au in comun caracteristici precum:
  - unitati lexicale (tokens)
  - sintaxa
  - semantica
- Diferente relativ la:
  - ambiguitate
  - redundanta
  - literalness: limbajul natural foloseste idiom-uri, metafore pe cand cel formal este exact
- Diferenta intre un limbaj natural si unul formal poate fi comparata cu diferenta intre poezie si proza.

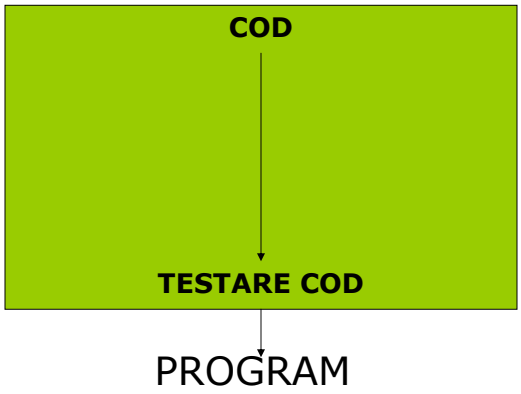
Cum se citeste un program?

- un limbaj formal (de programare) este mult mai dens) decat unul natural.
- structura unui program este foarte importanta: este gresit a se citi un program de la stanga la dreapta si de sus in jos
- incearca sa identifici token-urile si structura programului.
- detaliile sunt importante: o fraza in limbaj natural o intelegi si fara semne de punctuatie dar un program poate sa insemne altceva daca lipseste un semn.

Rezolvarea unei probleme



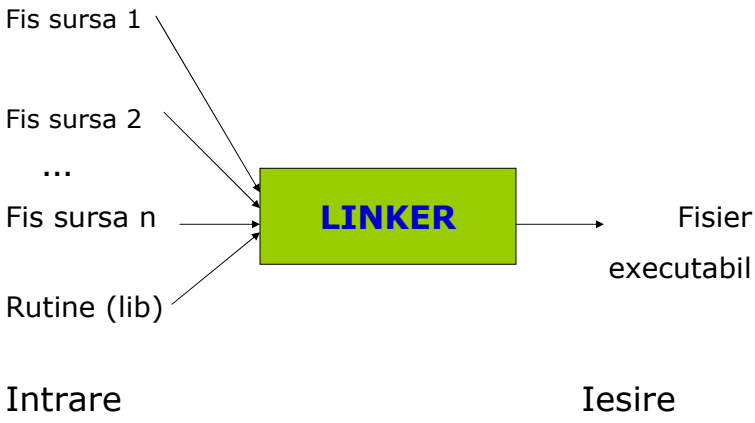
Implementarea



Compilator

Intrare Iesire

Editor de legaturi



String-matching , studiu de caz

- Intrare:
  - textul y = y[0..n-1] peste alfabetul Σ
  - pattern-ul x = x[0..m-1]
- Iesire:
  - aparitiile lui x in y
- Exemplu:

  y = bababaaaba (n = 10)
  x = aba (m = 3)
  Aparitiile lu x in y: 2, 4, 8 (sau 1, 3, 7)

  bababaaaba

Metoda (algoritmul brute-force):

- o “fereastra” de lungimea pattern-ului se pozitioneaza la stanga textului
- se compara caracterele din fereastra cu cele din pattern
- dupa ce se constata ca se potrivesc toate sau, dupa prima nepotrivire, se deplaseaza fereastra la dreapta cu un caracter si se reia procesul

String-matching , studiu de caz (2)

1. y babababaaaba
   1
   x aba
2. y bababababaaaba
   123
   x aba
3. etc.

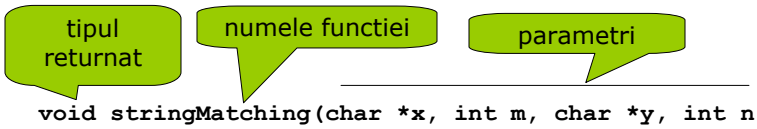
String-matching -Programul

```
#include <stdio.h>
void stringMatching(char *x, int m, char *y, int n){
   int i, j;
   for(j = 0; j <= n - m; ++j){
       for(i = 0; i < m && x[i] == y[i + j]; ++i);
       if(i >= m )
       printf(" %d ", j+1);
     }
}
int main(){
    char *a = "aba", *b = "bababaaaba";
    stringMatching(a, 3, b, 10);
    return 0;
}
```


String-matching –Programul disecat

```
#include <stdio.h>
```

- **directiva catre preprocesor**




```
void stringMatching(char *x, int m, char *y, int n)
```

- **linia de definitie (header-ul functiei):**
  - **tipul returnat: void – nu returneaza nici o valoare**
  - **numele functiei: este un identificator**
  - **parametrii functiei :x si y sunt pointeri la tipul char, deci siruri de caractere, m si n sunt intregi si reprezinta lungimile lui x repectiv y**
- **Corpul functiei este cuprins intre parantezele { }**

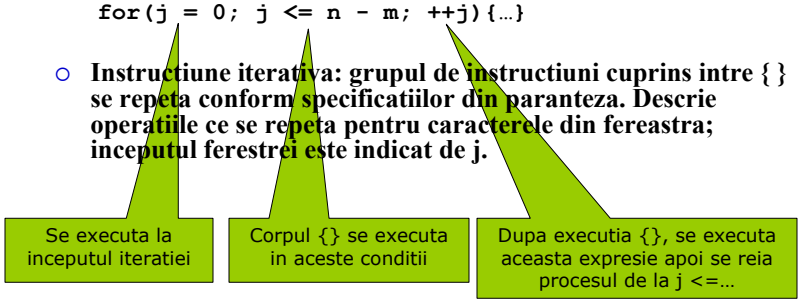
String-matching –Programul disecat

```
int i, j;
```

- **Declaratie pentru indicii din text repectiv pattern: i si j sunt declarate variabile de tip intreg**

```
for(j = 0; j <= n - m; ++j){…}
```

- **Instructiune iterativa: grupul de instructiuni cuprins intre { } se repeta conform specificatiilor din paranteza. Descrie operatiile ce se repeta pentru caracterele din fereastra; inceputul ferestrei este indicat de j.**


Se executa la inceputul iteratiei
Corpul {} se executa in aceste conditii
Dupa executia {}, se executa aceasta expresie apoi se reia procesul de la j <=…

String-matching –Programul disecat

```
for(i = 0; i < m && x[i] == y[i + j]; ++i);
```

- **Instructiune iterativa: delimitatorul `;` de la sfarsit arata ca aceasta iteratie are corpul vid**
  - **`i = 0;` expresia cu care incepe iteratia; testarea caracterelor din pattern incepe cu primul caracter**
  - **`i < m && x[i] == y[i + j];` in aceste conditii se continua cu expresia `++i` adica, daca se potrivesc caracterele din pattern cu cele din "fereastra" se continua testarea**

```
if(i >= m )
```

- **Instructiune de decizie: daca expresia din paranteza este diferita de zero atunci se executa instructiunea urmatoare lui if**

String-matching –Programul disecat

```
printf(" %d ", j+1);
```

- **Functie ce se afla in biblioteca – fisierul `stdio.h` si care are ca efect afisarea pe ecran (fisierul standard de iesire) a datelor indicate in argumente:**
  - **`" %d "` este argument sir de caractere care contine formatul de scriere**
  - **`j+1` este argument – expresie a carei valoare este scrisa pe ecran**

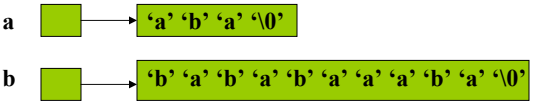
```
int main(){..}
```

- **Functia cu care incepe executia unui program C. Ea contine declaratii ale variabilelor – date de intrare, alte date, instructiuni, apel la alte functii, etc.**

String-matching –Programul disecat

```
char *a = "aba", *b = "babababaaaba";
```

- **Declaratie pentru datele de intrare:**
  - **`char*` este declaratie pentru pointer la tipul `char` – caracter**
  - **`a` este numele variabilei pointer**
  - **`"aba"` este o constanta sir de caractere. Declaratia are ca efect alocarea unei zone de memori de lungime 4 initializata cu `aba` (primii trei octeti) si caracterul special `'\0'` pentru ultimul octet. Variabila pointer `a` este initializata cu adresa primului octet alocat. In mod analog pentru b.**



String-matching -Varianta

```
#include <stdio.h>
#include <string.h>

void stringMatching(char *x, int m, char *y, int n){
     char *py;
     for(py = y; *y != '\0'; ++y)
        if(memcmp(x, y, m) == 0)
          printf(" %d ", y - py);
}

int main(){
    char *a = "aba", *b = "bababaaaba";
    stringMatching(a, 3, b, 10);
    return 0;
}
```


OBSERVATII

- Complexitatea timp (algoritmul brute-force) este O(m*x*n).
- Sunt cunoscuti mai mult de 30 algoritmi pentru rezolvarea acestei probleme
- Algoritmul Knuth-Morris-Prat are complexitatea O(n) dar are o faza de preprocesare ce necesita O(m*x*s) pasi, unde s este dimensiunea alfabetului

C - O trecere in revista
Un prim program C

```
#include <stdio.h>
int main(void)
{
  printf("Salut!\n");
  printf("Bine ati venit la cursul de
                Programare!");
}
```


Un prim program C

- Un program C este o succesiune de declaratii si definitii de functii; el specifica modul in care trebuie executat un calcul, fie el matematic sau simbolic
- Orice program C contine o functie cu nume impus **`main()`** cu care incepe executia sa
- O functie contine declaratii de variabile locale urmate de instructiuni (nu este permisa definirea de functii imbricate)

Expresii

- O variabila este numele unei locatii care memoreaza o valoare de un anumit tip(numar, sir de caractere etc.).
- Numele unei variabile este o succesiune de litere si cifre; acesta incepe cu o litera. Semnul _ (underscore) este o "litera".
- O expresie se constitue din variabile, operatori, apeluri de functii.
- Orice expresie are o valoare ce se determina dupa reguli ale limbajului.
- Valoarea expresie are un tip ce este determinat din tipul variabilelor si operatorii constituenti.
- Tipurile de baza(predefinite) in C sunt **`char int float double void`**. Cuvintele rezervate **`short`** si **`long`** influenteaza asupra domeniului de valori a tipului respectiv
- Un **`char`** este un intreg ce reprezinta codul ASCII al unui caracter

Expresii -continuare

- Expresia de atribuire:

  `variabila = expresie`

  are ca valoare valoarea expresiei din partea dreapta a semnului =

  i = 5

  a = b = c = 25

  max = a > b ? a : b

Declaratii

- Forma unei declaratii:

```
tip variabila;
tip var1, var2, …, varn;
tip variabila = expresie_constanta;
```

- Variabile globale: declararea lor se face la inceputul programului, in afara oricarei functii.
- Variabile locale: declararea se face in corpul functiei, la inceput.

Curs 2
Limbajul C – O trecere in revista

- Citire / scriere valori
- Adrese
- Instructiuni
- Tablouri
- Siruri de caractere
- Macrouri
- Structuri
- Fisiere

Citire / Scriere valori

```
/*Exemplul 1  printf() si scanf()*/
#include <stdio.h>
int  main(void){
  char    c1, c2;
  int     i;
  float   x;
  double  y;
  scanf("%c%c%d", &c1, &c2, &i);
  scanf("%f%lf", &x, &y);
  printf("%c%3c%5d\n", c1, c2, i);
  printf("%e%17.10e\n", x, y);
  return 0;
}
```

Citire / Scriere valori

```
/*Exemplul 2  printf() si scanf()*/
#include <stdio.h>
int  main(void){
  char    nume[20];
  printf("Care este numele tau?\n");
  scanf("%s", nume);
  printf("Salut, %s!\n", nume);
  return 0;
}
/*   Care este numele tau?
     Petrica Ionescu
     Salut, Petrica!     */
```


Adrese

- Orice variabila are o adresa in memorie
- Adresa variabilei x se obtine prin &x
- Un pointer **`p`** este o variabila ce are ca valoare o adresa (*tip* **`*p`**)
- Continutul memoriei de adresa p se obtine prin **`*p`**

Adrese - exemplu

```
/*Exemplul 3  adrese */
#include <stdio.h>
int  main(void){
  int x = 45;
  int *p = &x;
  printf("Valoarea lui x:%d\n", x);
  printf("Adresa lui x:%ld\n", &x);
  printf("Valoarea lui x:%d\n", *&x);
  printf("Valoarea lui x:%d\n", *p);
  printf("Adresa lui x:%p\n", p);
  return 0;
}
```


Adrese - rezultat

DJGPP Version 2.03 :

```
Valoarea lui x:45
Adresa lui x:588604
Valoarea lui x:45
Valoarea lui x:45
Adresa lui x:8fb3c
```

Visual C++ 6.0:

```
Valoarea lui x:45
Adresa lui x:6684148
Valoarea lui x:45
Valoarea lui x:45
Adresa lui x:0065FDF4
```

Instructiuni

- Instructiunea vida: `;`
- Instructiunea expresie:

```
a+b;    289;      d = b*b-4*a*c;
```

- Instructiunea compusa: {}

```
{a=b; b=c; c=a;}
```


Instructiunile `if`

```
if (expresie)
  instructiune

if (expresie)
  instructiune1
else
  instructiune2
```

Instructiunile `if` - exemplu

```
int max4(int i, int j, int k, int l){
    int max;
    if (i>j)
       if(k>l)
           if(i>k) max = i;
           else max = k;
       else
           if(i>l) max = i;
           else max = l;
    else
       if(k>l)
           if(j>k) max = j;
           else max = k;
       else
           if(j>l) max = j;
           else max = l;
       return max;
}
```


Operatorul ternar ?:

```
int Max4(int i, int j, int k, int l){
    return i>j ? k>l ? i>k ? i : k :
     i>l ? i : l : k>l ? j>k ? j : k :
     j>l ? j : l;
}
```

Instructiunea **switch**

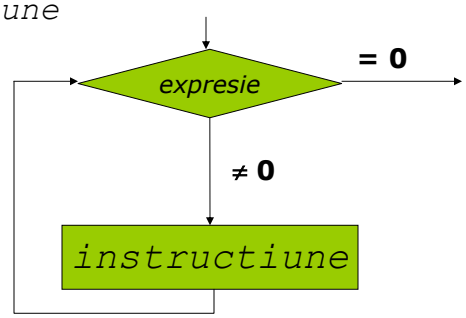
```
switch(expresie_int){
  case const1:
     instructiune1
  case const2:
     instructiune2
  …
  case constn:
     instructiunen
  default:
     instructiune
}
```


Instructiunea **switch**

```
#define BLANC ' '
#define VIRGULA ','
#define PUNCT '.'
#define RC '\n'
int main(){
   int nb = 0, nv = 0, np = 0, nl = 0, nc =0;
   char c;
   while ((c = getchar())!= EOF){
        nc++;
        switch (c){
                case BLANC : nb++; break;
                case VIRGULA : nv++; break;
                case PUNCT : np++; break;
                case RC: nl++;
        }
   }
/*…*/
}
```

Instructiunea **while**

while (*expresie*)
  instructiune



Instructiunea **while** - exemplu

```
#include <stdio.h>
int main(){
   int c;
   int blanks=0,tabs=0,newlines=0;
   printf("Introdu un text pe mai multe linii.\n");
   printf("La sfarsit apasa <Enter> apoi ^Z.\n");
   while ((c=getchar()) != EOF){
        if (c==' ') blanks++;
        if (c=='\t') tabs++;
        if (c=='\n')newlines++;
   }
   printf("No. of blanks:    %d\n",blanks);
   printf("No. of tabs:      %d\n",tabs);
   printf("No. of new lines: %d\n",newlines);
   return 0;
}
```

Instructiunea **do..while**

```
/* Suma numerelor multiple de 3 pana la N */
#include<stdio.h>
#define N 100
int main(){
   int i=1, suma=0;
   do{
       if(i%3 == 0) suma += i;
       i++;
   }while(i<=N);
   printf("suma = %d", suma);
   return 0;
}
/* 1683 */
```


Instructiunea **for**

```
for(expresie1; expresie2; expresie3)
  instructiune

expresie1;
while(expresie2){
  instructiune
  expresie3;
}
```

Instructiunea `for` - exemplu

```
#include <stdio.h>
#define N 30;
int main(void){
  int i, S = 0;
  for (i = 1; i <=N; ++i)
      S += i;
  printf("Suma primelor %d numere ");
  printf("naturale este: %d", S);
  return 0;
}
```


Explicatii

- Semnul `;` face parte din unele instructiuni. Trebuie sa apara si la sfarsitul:
  - declaratiilor
  - instructiunilor expresii
  - apelurilor de functii
  - instructiunilor `return, break, continue`
- Gruparea instructiunilor se face prin `{}`

Tablouri

```
#define N 20
int main(){
   int min = 0, max = 0, i, n;
   int vec[N];
   scanf(''%d'', &n);
   for (i=0; i<n; i++)
        scanf(''%d'', &vec[i]);
   if (n>0)
   min = max = vec[0];
   for (i=1; i<n; i++){
        if (vec[i] > max)
               max = vec[i];
        if (vec[i] < min)
               min = vec[i];
   }
/*…*/
}
```


Siruri de caractere

```
#define MAX_SIR 100
/*...*/
  char s[MAX_SIR];
  int i, j, lg;
  printf("Sirul: ");  scanf("%s", s);
  /* determina lungimea sirului */
  lg=0;
  while (s[lg] != '\0') lg++;
  /* testarea proprietatii de palindrom */
  i=0; j=lg-1;
  while (s[i]==s[j] && i<j)
       {++i; --j; }
  if (i >= j)
       printf("\nSirul este palindrom.\n");
  else
       printf("\nSirul nu este palindrom.\n");
  /*...*/
```

Macrodefinitii (macrouri)

```
#define sqr(x)    ((x)*(x))
#define abs(x)    ((x)>=0?x:-(x))
  a = sqr(2+b) + abs(u+v)
#define swap(t,a,b) {t x=a;a=b;b=x;}
   int i=10, j=64;
   float x=1.23,y=2.34;
   swap(int,i,j);
   swap(float,x,y);
```


Structuri

```
struct data{
   int zi, luna, an;
   char nume_zi[4];          /* lun, mar,… */
   char nume_luna[4];        /* ian, feb,… */
};
struct data azi, d1, zile_libere[8];

struct student{
   char *nume;
   char *prenume;
   int stud_id;
   int note[8];
};
typedef struct student student
student un_student={"Popescu", "Ion", "20030033"};
```

Structuri

```
typedef struct complex {
   double  real, imag;
}complex;


complex add(complex a, complex b){
   complex temp;
   temp.real = a.real + b.real;
   temp.imag = a.imag + b.imag;
   return temp;
}
```


Fisiere

- Declarare pointer catre fisier:

```
FILE *fin, *fout;
```

- Deschidere fisiere:

```
fin = fopen("un_fisier.txt","r");
fout = fopen("alt_fisier.txt","w");
```

- Citire din fisier:

```
while ((c = getc(fin)) != EOF)
   {/* prelucrare */}
fscanf("%d%f%c%s", &i,&x,&c,s);
```

- Inchidere fisier:

```
fclose(fin); fclose(fout);
```

Fisiere - Exemplu

```
/* Copiere fisier cu modificare litere mici */
#include <stdio.h>
#include <ctype.h>
#define   MAXSTRING   10
int main(void){
   char   file_name[MAXSTRING];   int    c;
   FILE   *ifp, *ofp;
   fprintf(stderr, "\nIntrodu numele unui fisier:  ");
   scanf("%s", file_name);
   ifp = fopen(file_name, "r");
   if(!ifp) { printf("Eroare!");  return 1;}
   ofp = fopen("test.out", "w");
   while ((c = getc(ifp))!=EOF) {
      if(islower(c)) c = toupper(c);
      putc(c, ofp);
   }
   close(ifp); close(ofp);
   return 0;
}
```


Argumente in main()
Linia de comanda cu argumente

- Functia main() cu argumente

```
int main(int nr_arg, char *arg[]) {
  /*... */
}
```

- Utilizarea argumentelor

```
finp = fopen(arg[1],"r");
fout = fopen(arg[2],"w");
```

- Linia de comanda cu argumente:

```
prog.exe fisier_intrare fisier_iesire
```

Argumente in main - Exemplu

```
#include <stdio.h>
#include <stdlib.h>
void   double_space(FILE *, FILE *);
void   prn_info(char *);
int main(int argc, char *argv[]){
   FILE   *ifp, *ofp;
   if (argc != 3) {prn_info(argv[0]);exit(1);}
   ifp = fopen(argv[1], "r");
   ofp = fopen(argv[2], "w");
   double_space(ifp, ofp);
   fclose(ifp);
   fclose(ofp);
   return 0;
}
```


Argumente in main - Exemplu

```
void double_space(FILE *ifp, FILE *ofp){
   int  c;
   while ((c = getc(ifp)) != EOF) {
      putc(c, ofp);
      if (c == '\n')
       putc('\n', ofp);   /* se adauga o linie noua
   */
   }
}

void prn_info(char *prog_name){
   printf("\n%s%s%s\n\n%s%s\n\n",
      "Utilizati:  ", prog_name, "  infile
   outfile",
      "Continutul fisierului infile va fi dublu
   spatiat    si rescris in fisierul outfile.");
}
```

Glosar

- variabila
- valoare
- tip
- cuvant cheie
- instructiune (statement)
- declaratie
- asignare
- expresie
- operator
- operand
- precedenta
- concatenare
- compunere (instructiune compusa)

Curs 3

- Caracterele limbajului C
- Cuvinte rezervate, reguli de sintaxa
- Tipuri de date fundamentale
  - Domeniul
  - Operatii
  - Constante
  - Functii
- typedef

Caracterele limbajului C

- Litere:

A B C D … X Y Z

a b c d … x y z

- Cifre: 0 1 2 3 4 5 6 7 8 9
- Alte caractere:

+ - * / = ( ) { } [ ] < > ` " ! # % & _

|^ ~ \ . , ; : ?

- Caractere spatiu: blank, newline,tab, etc.

Reguli de sintaxa – Forma BNF

litera_mare ::= A | B | … | Z
litera_mica ::= a | b | … | z
litera ::= *litera_mare* | *litera_mica* | _
cifra_pozitiva ::= 1| 2 | … | 9
cifra ::= 0 | *cifra_pozitiva*
identificator ::=
    litera | *identificator litera*
    |*identificator cifra*
instr_if ::= if (*expr*) *instr* {else *instr*}$_{opt}$

Cuvintele cheie (rezervate)

auto	**double**	**int**	**struct**
break	**else**	**long**	**swich**
case	**enum**	**register**	**typedef**
char	**extern**	**return**	**union**
const	**float**	**short**	**unsigned**
continue	**for**	**signed**	**void**
default	**goto**	**sizeof**	**volatile**
do	**if**	**static**	**while**

Tipuri de date

- Tip de date :
  - Entitati de tip data
  - Domeniu de valori
  - Operatii ce se pot efectua cu aceste date
- Tipuri de date
  - predefinite – tipuri fundamentale
  - definite de utilizator (programator)

Tipuri de date fundamentale

```
char        signed char      unsigned char
signed short int      unsigned short int
signed int            unsigned int
signed long int       unsigned long int
float       double       long double
```

ECHIVALENTE

```
signed short int     ≡     short
unsigned short int   ≡     unsigned short
signed int           ≡     int
unsigned int         ≡     unsigned
signed long int      ≡     long
unsigned long int    ≡     unsigned long
```

Tipul intreg

- **int**
  sizeof(int) = 2 sau **4** octeti
- **short int** sau **short**
  sizeof(short)=2, {-32768,…,32767}
- **long int** sau **long**
  sizeof(long) = 4
  {-2 147 483 648, …, 2 147 483 647}
- **signed int, signed short int, signed long int**

Intregi fara semn

- **unsigned int**
- **unsigned short int**
  {0,…,65535}
- **unsigned long int**

- Nu exista overflow (depasire) – calculul se efectueaza modulo 2^n, unde n este numarul de biti

Intregi “foarte scurti”: **`char`**

- Tipul **`char`** este o submultime a tipului **`int`**
- **`char`** reprezinta, in functie de masina, domeniul de valori:
  {-128, …, 127} sau {0,…,255}
- **`unsigned char`** {0,…,255}
- **`signed char`** {-128, …, 127}

`sizeof(char) = 1`

Constante intregi

- Octale : au prefixul 0 (zero)
  032 = 26 077 = 63
- Hexazecimale: au prefixul 0x sau 0X
  0x32 = 50 0x3F = 63
- Intregi “long”: au sufixul l sau L
  2147483647L, 0xaf9Fl = 44959
- Intregi “unsigned” au sufixul u sau U
  345u 0xffffu = 65535

Constante caracter

- Caractere intre apostrof: ‘A’, ‘+’, ‘n’
- Caractere in zecimal: 65, 42
- Caractere in octal: ’\101’, ‘\52’
- Caractere in hexazecimal: ‘\x41’, ‘\x2A’
- Notatii pentru caractere speciale: ‘\n’, ‘\t’, ‘\r’, ‘\\’, ‘\a’, ‘\”, ‘\0’, ‘\”’

Constante caracter - exemplu

```
/* Exemple de constante caracter */
#include <stdio.h>
int main(){
    char a, b, c, d;
    a = 'A'; b = 65; c = '\101'; d = '\x41';
    printf("%c %c %c %c\n", a, b, c, d);
    printf("%c %d %o %x\n", a, a, a, a);
    return 0;
}
/*
A A A A
A 65 101 41
*/
```

Codurile ASCII

```
# include <stdio.h>
int main (void){
  short c;
  for(c=0; c<= 127; c++){
    printf("cod ASCII:%d",c);
    printf(" caracter:%c\n",c);
  }
  return 0;
}
/* for(c='a'; c<='z'; c++) */
```


Macrourile getchar() si putchar()

- Sunt definite in **<stdio.h>**
- Citire caracter de la tastatura
- Scriere caracter pe ecran

```
#include <stdio.h>
int main(void){
  char c;
  while ((c=getchar()) != EOF)
   {putchar(c); putchar(c);}
  return 0;
}
```

Operatii, Functii

- Operatii pentru tipurile intregi:

+ - * / %

== != < <= > >=

++ -- (aplicate doar variabilelor)

- Functii:
  - cele de la tipul flotant
  - cele din biblioteca **<ctype.h>: tolower, toupper, isalpha, isalnum, iscntrl, isdigit, isxdigit, islower, isupper, isgraph, isprint, ispunct, isspace**

Operatorii ++ si --

- Se aplica doar unei expresii ce desemneaza un obiect din memorie (L-value), deci doar variabilelor:

Expresie	++i	i++	--i	i--
Valoare	i+1	i	i-1	i
i dupa evaluare	i+1	i+1	i-1	i-1

++5 --(k+1) ++i++ nu au sens

Tipul flotant (real)

- **`float`**
  - Numere reale in simpla precizie
  - **`sizeof(float)`** = 4
  - -10^{-37} <= abs(f) <= 10^{38}
  - 6 cifre semnificative
- **`double`**
  - Numere reale in dubla precizie
  - **`sizeof(float)`** = 8
  - -10^{-307} <= abs(f) <= 10^{308}
  - 15 cifre semnificative

Tipul flotant (real)

- **`long double`**
  - Numere reale in "extra" dubla precizie
  - **`sizeof(float)`** = 12
  - -10^{-4931} <= abs(f) <= 10^{4932}
  - 18 cifre semnificative
- Operatii:

`+ - * /`

`== != < <= > >=`

Constante reale

- Constantele reale sunt implicit **`double`**

125.435 1.12E2 123E-2 .45e+6 13. .56

- Pentru a fi **`float`** trebuie sa aiba sufixul f sau F

.56f 23e4f 45.54E-1F

- Pentru long double trebuie sa aiba sufixul l sau L 123.456e78L

Functii (in biblioteca math.h)

sin cos tan asin acos
atan sinh cosh tanh exp
log log10 pow sqrt ceil floor
fabs ldexp frexp modf fmod

Exemplu utilizare functii

```
#include <math.h>
#include <stdio.h>
int main(void){
   double x;
   while(1){
       printf("Introdu x: ");
       if (scanf("%lf", &x) != 1) break;
       if (x >= 0.0)
         printf("%22.15e %22.15e %22.15e\n",
                    x, sqrt(x), pow(x,x));
       else{
         printf(" \n Numar negativ!\n");
         break;
       }
   }
}
```


Utilizare `typedef`

- Mecanism prin care se asociaza un tip unui identificator:

```
typedef char   litera_mare;
typedef short  varsta;
typedef unsigned long   size_t;
```

- Identificatorul respectiv se poate utiliza pentru a declara variabile sau functii:

```
litera_mare    u, v='a';
varsta         v1, v2;
size_t         dim;
```

Date booleene (logice)

- Nu exista un tip special pentru date logice
- Domeniul de valori: {false, true}
- false = 0
- true = orice intreg nenul
- Operatii:

! && || == !=

- O declaratie posibila:

```
typedef enum {false = 0, true = 1} bool;
bool x, y;
```


Expresii logice

expresie_relationala ::=

expr < *expr* | *expr* > *expr*

| *expr* <= *expr* | *expr* >= *expr*

| *expr* == *expr* | *expr* != *expr*

expresie_logica ::= ! *expr*

| *expr* || *expr*

| *expr* && *expr*

Valoarea expresiilor relationale

a-b	a<b	a>b	a<=b	a>=b	a==b	a!=b
pozitiv	0	1	0	1	0	1
zero	0	0	1	1	1	0
negativ	1	0	1	0	0	1

Valoarea expresiilor logice ||

exp1	exp2	exp1 || exp2
≠ 0	Nu se evalueaza	1
=0	Se evalueaza	1 daca exp2 ≠ 0 0 daca exp2 = 0

Valoarea expresiilor logice &&

exp1	exp2	exp1 && exp2
= 0	Nu se evalueaza	0
≠ 0	Se evalueaza	1 daca exp2 ≠ 0 0 daca exp2 = 0

Exemple

- O conditie de forma a ≤ x ≤ b se scrie in limbajul C:

```
(x >= a) && (x <= b)   sau
a <= x && x <= b
```

- O conditie de forma a > x sau x > b se scrie in limbajul C:

```
x < a || x > b      sau
!(x >= a && x <= b)
```

Curs 4

- Operatori de atribuire
- Tipul void
- Operatorul **`sizeof`**
- Precedenta operatorilor
- Reguli pentru conversia implicita
- Fortarea tipului – cast
- Fisiere .h relative la tipuri
- Instructiuni

Operatorul conditional `?:`

exp1 `?` *exp2* `:` *exp3*

- Se evalueaza *exp1*
- Daca *exp1* are valoare nenula (true) atunci valoarea expresiei este valoarea lui *exp2*; *exp3* nu se evalueaza
- Daca exp1 are valoare nula (false) atunci valoarea expresiei este valoarea lui *exp3*; *exp2* nu se evalueaza
- Operatorul `?:` este drept asociativ

Operatorul conditional ? : Exemple

```
x >= 0 ? x : y
x > y ? x : y
x>y ? x>z ? x :z : y>z ? y : z
```

```
#include <stdio.h>
int main(void){
    int a=1, b=2, c=3;
    int x, y, z;
    x = a?b:c?a:b;
    y = (a?b:c)?a:b; /* asociere stanga */
    z = a?b:(c?a:b); /* asociere dreapta */
    printf("x = %d, y = %d, z = %d\n", x, y, z);
}
 /* x = 2, y = 1, z = 2 */
```


Operatorul "=" (atribuire)

- Expresia de atribuire:

  exp1 = exp2

- *exp1* este o "L-value" (obiect din memorie: variabila, variabila tablou cu indici)
- Tipul expresiei este tipul lui *exp1*
- Se evalueaza *exp2* apoi *exp1* capata valoarea lui *exp2* eventual converita.
- Asadar, operatorul = modifica valoarea operandului stang
- Valoarea expresiei este valoarea lui *exp1* dupa evaluare

Operatorul “=“ (atribuire)

- Exemple:

  x = sqrt(9)
  a = (b = 2) + (c = 3)
  a = b = c = 0 echivalenta cu
  a = (b = (c = 0))
  while((c = getchar()) != EOF) putchar(c);

- Nu confundati *e1 = e2* cu *e1 == e2*

```
a = 22;
if ( a == 0)printf("nul")
else printf("nenul");        /* nenul */

if ( a = 0)printf("nul")
else printf("nenul");        /* nul */
```


Operatori de atribuire compusa

- O expresie de atribuire compusa are forma:

exp1 op= exp2

unde *op=* este unul din:

`+=  -=  *=  /=  %=  &=  |=  ^=  >>=  <<=`

- Expresia este echivalenta cu

exp1 = exp1 op (*exp2*)

cu precizarea ca *exp1* se evalueaza o singura data.

`j *= k + 3` echivalenta cu: `j = j*(k+3)`

`j *= k = m + 5` echivalenta cu:

```
j = (j * (k = (m + 5)))
```

Operatorul virgula ,

expresia_virgula ::= *expresie*, *expresie*

- Se evalueaza prima expresie apoi cea de-a doua.
- Valoarea si tipul intregii expresiii este valoarea si tipul operandului drept.
- Operatorul virgula are cea mai mica precedenta.

- Exemple:

```
a = 1, b = 2
i = 1, j = 2, ++k + 1
k != 1, ++x * 2.0 + 1
for(suma = 0, i = 1; i <= n; suma += i, ++i);
```


Tipul void

- Conversia in tip **`void`** a unei expresii semnifica faptul ca valoarea sa este ignorata
- Utilizat pentru tipul pointer; nu se face controlul tipului la un pointer de tip **`void`**
- Utilizat pentru functii fara valoare returnata sau pentru functii fara parametri

Operatorul `sizeof()`

- Operator unar ce permite gasirea numarului de octeti pe care se reprezinta un obiect (tip, expresie)

```
sizeof(int), sizeof(double)
sizeof(b*b-4*a*c), sizeof(i)
```

sizeof(char)<=sizeof(short)<=sizeof(int)<=sizeof(long)
sizeof(signed)=sizeof(unsigned) = sizeof(int)
sizeof(float) <= sizeof(double) <= sizeof(long double)

Operatorul `sizeof()`

```
#include<stdio.h>
int main(void){
  int x=1; double y=9; long z=0;
  printf("Operatorul sizeof()\n\n\n");
  printf("sizeof(char)         = %2u\n",sizeof(char));
  printf("sizeof(int)          = %2u\n",sizeof(int));
  printf("sizeof(short)        = %2u\n",sizeof(short));
  printf("sizeof(long)         = %2u\n",sizeof(long));
  printf("sizeof(float)        = %2u\n",sizeof(float));
  printf("sizeof(double)       = %2u\n",sizeof(double));
  printf("sizeof(long double)  = %2u\n",sizeof(long
   double));
  printf("sizeof(x +y + z)     = %2u\n",sizeof(x+y+z));
  printf("sizeof(void)         = %2u\n",sizeof(void));
  return 0;
}
```

Operatorul `sizeof()`

Rezultatul executiei Visual C++ (Djgpp):

```
sizeof(char)        =  1
sizeof(int)         =  4
sizeof(short)       =  2
sizeof(long)        =  4
sizeof(float)       =  4
sizeof(double)      =  8
sizeof(long double) =  8  ( 12 Djgpp)
sizeof(x +y + z)    =  8
sizeof(void)        =  0  (1 Djgpp)
```


Precedenta operatorilor

OPERATORI	**ASOCIERE**
() ++ -- (postfix)	stânga
++ -- *(prefix)* ! & *(adresa)* *** *(deref)* + - *(unari)* sizeof()**	dreapta
* / %	stânga
+ -	stânga
< <= > >=	stânga
== !=	stânga
&&	stânga
||	stânga
?:	dreapta
= += -= *= /= %=	dreapta
, **(operatorul virgula)**	stânga

Compatibiltatea tipurilor predefinite

- Toate tipurile aritmetice: intregi, caractere, reale sunt compatibile intre ele; compilatorul admite orice combinatie de obiecte de aceste tipuri intr-o expresie
- La evaluarea unei expresii in care apar tipuri diferite compilatorul face conversii implicite

Reguli pentru conversia implicita

- In absenta unui **`unsigned`** obiectele se convertesc la tipul cel mai "inalt" in ordinea (descrescatoare):

`long double, double, float, long int,int`

- Regulile pentru operanzii **`unsigned`** depind de implementare.
- Conversia la **`unsigned`** se face doar in caz de necesitate (de ex. valoarea din **`unsigned`** nu "incape" in celalat operand).

Reguli pentru conversia implicita

- Regula "integer promotion" : operatiile se fac cel putin in **`int`**, deci **`char`** si **`short`** sunt "promovati" la **`int`**.
- La o asignare (v = exp) tipul membrului drept se converteste la tipul membrului stang(care este si tipul rezultatului).
  - ATENTIE: se pot produce
    - Pierderea preciziei (double ->float ->long int)
    - Pierderea unor biti semnificativi (long ->int)
    - Nedeterminari

Exemplu:

```
/*Exemplu asignari */
#include <stdio.h>
int  main(void){
     char c1 = -126, c2; /*  c1 = 10000010 */
     unsigned char c3,  c4 = 255; /* c4 = 111111111 */
     short s1, s2 = -32767; /* s2=10000000 00000001 */
     short s3 = -1, s4; /* s3 = 11111111 11111111 */
     s1 = c1;
     printf("c1 = %d, s1 = %d\n", c1, s1);
     c2 = s2;
     printf("c2 = %d, s2 = %d\n", c2, s2);
     c3 = s3;
     printf("c3 = %d, s3 = %d\n", c3, s3);
     s4 = c4;
     printf("c4 = %d, s4 = %d\n", c4, s4);
     return 0;
}
```

Exemplu(rezultatul executiei):

```
c1 = 10000010                s1 = c1;
c1 = -126, s1 = -126
s2=10000000 00000001         c2 = s2;
c2 = 1, s2 = -32767
s3 = 11111111 11111111       c3 = s3;
c3 = 255, s3 = -1
c4 = 111111111               s4 = c4;
c4 = 255, s4 = 255
```


Fortarea tipului - cast

- Conversia explicita la tipul *numetip*:
  (*numetip*) *expresie*
- Exemple:

```
(long)('A' + 1.0)
(int)(b*b-4*a*c)
(double)(x+y)/z
(float)x*y/z
x/(float)2
```

Exemplu cast

```
#include <stdio.h>
int main(void){
  int i, j; double x, y, z, t;
  i=5/2; x=5/2; y=(double)(5/2);
  j=(double)5/2; z=(double)5/2;
  t=5./2;
  printf("%d, %g, %g, %d, %g, %g\n",
           i,  x,  y,  j,  z,  t);
}
/* 2, 2, 2, 2, 2.5, 2.5  */
```


Fisiere in biblioteca relative la tipuri

- <limits.h> - pentru tipurile de baza
  - Intregul min/max: INT_MIN, INT_MAX
  - Numarul de biti pe caracter CHAR_BIT
  - Etc.
- <float.h> - pentru tipurile flotante:
  - Exponentul maxim
  - Precizia zecimala, etc.
- <stdlib.h> - contine functii de conversie:
  - Sir de caractere in `int` : `atoi(const char*)`
  - Sir de caractere in `float`:`atof(const char*)`

Instructiuni

- Expresii: ; *expresie*;
- Compuse (bloc): {*declaratii instructiuni*}
- Conditionale: *if* *if-else* *switch-case*
- Iterative: *for* *while* *do-while*
- Intreruperea secventei:
  `continue; break; return expr;`
- Salt neconditionat: **`goto`**

Instructiunea expresie

instr_expresie ::= {*expresie*}$_{opt}$;

- Exemple:

```
a = b;
a + b + c;
;
printf("a= %d\n", a);
scanf("%d%f%c", i, x, c);
```

Instructiunea compusa (bloc)

instr_compusa ::=

{ $\{$*lista_declaratii*$\}_{0+}\{$*lista_instructiuni*$\}_{0+}$ **}**

- Grupeaza instructiuni intr- ounitate executabila.
- Daca sunt si declaratii la inceput, instructiunea compusa se numeste si *bloc.*
- O instructiune compusa este ea insasi o instructiune: oriunde se poate pune o instructiune, este corect sa se puna si o instructiune compusa.

Instructiunea compusa - Exemple

```
{
   a += b += c;
   printf("a = %d, b = %d, c = %d\n, a, b, c);
}
if(x > y){
   int temp;
   temp = x; x = y; y = temp;
}
{
   int a, b, c;
   {
        b = 2;c = 3;a = b += c;
   }
   printf("a= %d", a);
}
```

Instructiunile conditionale if si if-else

instr_if ::= **`if (`***expr***`)`** *instructiune*

instr_if-else ::= **`if (`***expr***`)`** *instructiune*
`else` *instructiune*

expr este o conditie construita cu:

- Expresii aritmetice
- Comparatori: ==, !=, <, <=, >, >=
- Conectori logici: &&, ||, !

Instructiunile conditionale if si if-else

Exemple:

```
if(b == a) aria = a*a;

if(x < y)
   min = x;
else
   min = y;

if(a%2) if(b%2) p = 1; else p = 2;

if(a%2){
   if(b%2) p = 1;
}else p = 2;
```

Instructiunile if si if-else - Exemple

```
if (i>j)
   if(k>l)
       if(i>k) max = i;
       else max = k;
   else
       if(i>l) max = i;
       else max = l;
else
   if(k>l)
       if(j>k) max = j;
       else max = k;
   else
       if(j>l) max = j;
       else max = l;
```


"Dangling else Problem"

```
if (a == 1)
     if (b == 2)
          printf("*****\n");
else
     printf("ooooo\n");
```

- Nu lasati forma codului sa va duca in eroare! Regula este: **`else`** este atasat celui mai apropiat **`if`**.

If-else-exemplu

```
int main(void){
    float x, y, rezultat;
    char operator;
    printf("Expresia:(numar operator numar)\n");
    scanf("%f %c %f", &x, &operator, &y);
    if(operator == '+')
          rezultat = x+y;
    else if(operator == '-')
          rezultat = x-y;
    else if(operator == '*')
          rezultat = x*y;
    else if(operator == '/')
          rezultat = x/y;
    else{
          printf("Eroare in scrierea expresiei!");
          return 1;
    }
    printf("Rezultatul este: %f\n", rezultat);
    return 0;
}
```


Instructiunea switch

```
switch(expresie_intreaga){
  case exp_const1: instr1
  case exp_const2: instr2
  …
  case exp_constn: instrn
  default: instructiune
}
```

Instructiunea switch

- Valoarea expresiei *expresie_intreaga,* care este de tip **`int`**, se compara cu constantele *exp_const.*
- In caz de egalitate se executa instructiunea corespunzatoare si toate cele ce urmeaza. Exista posibilitatea de iesire cu instructiunea **`break`**.
- Daca valoarea determinata difera de oricare din constantele specificate, se executa instructiunea specificata la **`default`**, care apare o singura data nu neaparat la sfarsit. Daca **`default`** lipseste se iese din **`switch`**.
- Valorile constantelor trebuie sa fie diferite; ordinea lor nu are importanta.
- Acoladele ce grupeaza multimea **`case`**-urilor sunt obligatorii. Dupa fiecare **`case`** pot apare mai multe instructiuni fara a fi grupate in acolade.

Instructiunea switch - exemple

```
scanf("%d", &i);
switch(i){
   case 1: printf(" 1");
   case 2: printf(" 2");
   case 3: printf(" 3");
   case 4: printf(" 4");
   default: printf(" blabla!");
}
2
   2 3 4 blabla!
```

Instructiunea switch - exemple

```
scanf("%d", &i);
switch(i){
   case 1: printf(" 1"); break;
   case 2: printf(" 2"); break;
   case 3: printf(" 3"); break;
   case 4: printf(" 4"); break;
   default: printf(" blabla!");
}
2
   2
```


Instructiunea switch - exemple

```
#include <stdio.h>
int main(void){
   float x, y, rezultat;
   char operator;
   printf("Expresia:(numar operator numar)\n");
   scanf("%f %c %f", &x, &operator, &y);
   switch(operator){
     case '+': rezultat = x+y; break;
     case '-': rezultat = x-y; break;
     case '*': rezultat = x*y; break;
     case '/': if(y){ rezultat = x/y; break; }
               else { printf("Impartire prin zero!");return 1;}

     default : printf("Eroare in scrierea expresiei!");return 1;
   }
   printf("Rezultatul este: %f\n", rezultat);
   return 0;
}
```

Instructiunea while

instructiunea_while ::=
`while`(*expresie*) *instructiune*

`while` (*expresie*){
instructiune
}
instructiunea_urmatoare

- Se evalueaza *expresie:* daca valoarea sa este nenula se executa *instructiune* si controlul este transferat inapoi, la inceputul instructiunii while; daca valoarea este nula se executa *instructiunea_urmatoare.*
- Asadar *instructiune* se executa de zero sau mai multe ori.

Instructiunea while

```
while (i++ < n)
  factorial *= i;

while((c = getchar() != EOF){
  if(c >= 'a' && c <= 'z')
     ++ contor_litere_mici;
  ++contor_total;
}
```

Instructiunea do..while

instructiunea_do..while ::=
do *instructiune* **while**(*expresie*);

```
do{
        instructiune
} while (expresie);
instructiunea_urmatoare
```

- Se executa *instructiune.*
- Se evalueaza *expresie:* daca valoarea sa este nenula controlul este transferat inapoi, la inceputul instructiunii do..while; daca valoarea este nula se executa *instructiunea_urmatoare.*
- Asadar *instructiune* se executa odata sau de mai multe ori.

Instructiunea do..while

```
do{
  c = getchar();
} while(c == ' ');

do{
  printf("Introdu un intreg pozitiv:")'
  scanf("%d", &n);
  if(error = (n <= 0))
     printf("\nEroare! Maiincearca!\n");
}while(error);
```

Exemplu - calculator

```
#include <stdio.h>
int main(void){
    float x, y, rezultat;
    char operator, c;
    int ERROR;
    printf("Calculator pentru expresii de forma
                numar operator numar\n");
    printf("Folositi operatorii + - * / \n");
```


Exemplu - calculator

```
do{
      ERROR = 0;
      printf("Expresia: ");
      scanf("%f %c %f", &x, &operator, &y);
      switch(operator){
        case '+': rezultat = x+y; break;
        case '-': rezultat = x-y; break;
        case '*': rezultat = x*y; break;
        case '/': if(y != 0) rezultat = x/y;
                  else { printf("Impartire prin zero!\n");ERROR = 1;}
                  break;

        default : {printf("Operator necunoscut!\n");ERROR = 1;}
      }
      if(!ERROR)
        printf("%f %c %f = %f \n", x, operator, y, rezultat);
      do{ printf("Continuati (d/n)?"); c = getchar();
      } while (c != 'd' && c != 'n');
  } while (c != 'n');
  printf("La revedere!\n");
  return 0;
}
```

Curs 5

- Instructiuni (continuare)
  - for, continue, break, return, goto
- Pointeri
- Functii
  - Definitie
  - Invocare: "apel prin valoare"
  - Pointeri ca parametri – efectul "apel prin referinta"
  - Functii cu numar variabil de parametri

Instructiuni

- Expresii: ; *expresie*;
- Compuse (bloc): {*declaratii instructiuni*}
- Conditionale: *if* *if-else* *switch-case*
- Iterative: *while* *do-while* *for*
- Intreruperea secventei:
  `continue; break; return expr;`
- Salt neconditionat: **`goto`**

Instructiunea for

instructiunea_for ::=
`for` (*expr1; expr2; expr3*) *instructiune*

`for` (*expr1; expr2; expr3*){
instructiune
}
instructiunea_urmatoare

- Una, doua sau toate trei dintre expresii pot lipsi, dar cei doi separatori sunt obligatorii.

Instructiunea for

- Daca *instructiune* nu contine **`continue`** si *expr2* este prezenta, atunci **`for`** este echivalent cu:

```
expr1;
while(expr2){
instructiune
expr3;
}
instructiunea_urmatoare
```

- Daca exista **`continue`** atunci aceasta transfera controlul la *expr3.*

Instructiunea for

- Se evalueaza *exp1* - in general aceasta se utilizeaza pentru initializarea iteratiei.
- Se evalueaza *exp2* - in general aceasta este o expresie logica ce se utilizeaza pentru controlul iteratiei. Daca valoarea sa este nenula(true), se executa corpul buclei do (instructiune), se evalueaza *exp3* si controlul este trecut la inceputul buclei do, fara a se mai evalua *exp1*.
- In general *exp3* face trecerea la iteratia urmatoare: modifica o variabila ce intra in componenta lui *exp2.*
- Procesul continua pana cand valoarea *exp2* este nula (false). Controlul este transferat urmatoarei instructiuni(cea de dupa for).

Instructiunea for - exemplu

```
#define N 100
int i, suma=0;
for(i = 1; i<=N; i++)suma+=i;

int suma, i;
for(suma = 0, i=0; i <= N; suma += ++i);

int i = 0;
char c;
for(; (c = getchar()) != '\n'; ++i)
     putchar(c);
```

Instructiunea for - exemplu

```
i = 1;
suma = 0;
for(;i <= N;++i) suma += i;

i = 1;
suma = 0;
for(;i <= N;) suma += i++;

i = 1;
suma = 0;
for(;;) suma += i++; // Bucla infinita
```


Instructiunea for - exemplu

```
/* Ce se afiseaza la executia programului? */
#include<stdio.h>
void main(){
int a, b, i, n = 5;
for (i = 3, a = b = 1; i <= n; a = b + (b = a), i++);
      printf("%d ", a);
}
/*
      a)      4
      b)      5
      c)      8
      d)      1 2 4
      e)      1 3 5
      f)      eroare la compilare
*/
```

Instructiuni de intrerupere a secventei

- **`continue;`**
  - se refera la bucla(for, while, do..while) cea mai apropiata.
  - intrerupe executia iteratiei curente si trece controlul la iteratia urmatoare.
- **`break;`**
  - se refera la bucla(for, while) sau instr. switch cea mai apropiata.
  - produce iesirea din bucla sau din switch si trece controlul la instructiunea urmatoare
- **`return expr;`** *sau* **`return;`**
  - in functii, intrerupe executia si transfera controlul apelantului, eventual cu transmiterea valorii expresiei *expr.*

Exemplu – for..continue

```
/* Suma numerelor multiple de 3 pana la N */

#include<stdio.h>
#define N 100
int main(){
int i, suma=0;
for(i = 1; i<=N; i++){
  if(i%3 != 0) continue;
  suma+=i;
}
printf("suma = %d", suma);
return 0;
}
/* suma = 1683 */
```

Exemplu – while..continue

```
/* Suma numerelor multiple de 3 pana la N */
#include<stdio.h>
#define N 100
int main(){
int i=0, suma=0;
while(i<=N){
   i++;
   if(i%3 != 0) continue;
   suma+=i;
}
printf("suma = %d", suma);
return 0;
}
/* suma = 1683 */
```


Exemplu – for – continue - break

```
#include <stdio.h>

int main(void){
  for(putchar('1');putchar('2');putchar('3')){
     putchar('4');
     continue;
     // break;
     putchar('5');
  }
  return 0;
}
```

continue vs. break

```
int t;
for(;;){
  scanf("%d",&t);
  if(t==0)continue;
  printf("%d\t",t);
}
```

```
int t;
for(;;){
  scanf("%d",&t);
  if(t==0) break;
  printf("%d\t",t);
}
```


Instructiuni de iteratie - recomandare

- Pentru expresia de control a iteratiei se recomanda operatorii relationali in locul celor de (ne)egalitate:

```
int contor = 0;
double suma = 0.0, x;
for(x = 0.0; x != 9.9; x += 0.1){
  suma += x; ++cnt;
}
printf("suma = %f, contor = %d\n");
return 0;
}
```

- Folositi `x < 9.9` in loc de `x != 9.9`

Instructiuni de intrerupere a secventei

goto *identificator;*

…

identificator : *instructiune*

- *identificator* este o eticheta.
- utilizarea acestui identificator este limitata: el reprezinta doar eticheta instructiunii tinta
- nu se poate face salt dintr-o functie in alta
- se recomanda utilizarea instructiunii **goto** numai in cazuri exceptionale

Exemplu – goto

```
#include <stdio.h>
#include <limits.h>
int main(void){
    //int a = 455, b = 17;
    int a = 3, b = INT_MAX - 2;
    //int a = -6, b = INT_MIN + 5;
    int c, d, cod_err = 0;
    if (b==0)
       { cod_err = 1; goto eroare; }
    else c = a/b;
    if ((b > 0) && (a > INT_MAX - b))
       { cod_err = 2; goto eroare; }
    else if ((b < 0) && (a < INT_MIN - b))
       { cod_err = 3; goto eroare; }
    else  d = a+b;
```

Exemplu – goto

```
eroare:
    switch (cod_err){
     case 0: printf(" c = %d, d = %d\n",c, d);break;
     case 1: printf(" Eroare: impartire prin zero!\n");
             break;
     case 2: printf(" Eroare: Depasire superioara!\n");
             break;
     case 3: printf(" Eroare: Depasire inferioara!\n");
             break;
    }
    return 0;
}
```


Instructiuni de control - rezumat

- Expresiile relationale si cele logice au ca valori intregii 1 sau 0;
- `!!a` si `a` pot avea valori distincte;
- Expresiile relationale si cele logice sunt utilizate pentru controlul fluxului; atentie la conversiile implicite;
- Intr-o instructiune if-else partea else este asociata celui mai apropiat if. Utilizati {} pentru eliminarea confuziilor;
- Executia iterativa a unei parti de cod se poate face folosind for, while, do..while. Atentie la expresia ce controleaza iteratia.
- Transferul neconditionat al controlului se obtine prin una din instructiunile goto, break, continue, return.

Pointeri

- Daca v este o variabila, &v este locatia sau adresa din memorie unde este pastrata valoarea sa.
- Operatorul adresa & ; precedenta, asociativitate.
- Adresele sunt valori intregi pozitive ce sunt interpretate ca adrese, inclusiv adresa speciala 0 (NULL).
- Variabilele pointer sunt variabile ce au ca valori adrese de memorie.

Pointeri

- Declararea unei variabile pointer:
  *tip *nume_var_pointer*;
- *nume_var_pointer* este o variabila ce poate avea valori adrese din memorie ce contin valori de tip *tip.*
- Exemple:

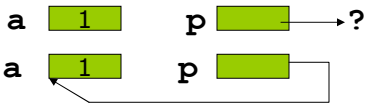
```
int *p, i;
p = 0;
p = NULL;
p = &i;
p = (int*) 232;
```

- p "pointeaza la i", "contine adresa lui i", "referentiaza la i".

Pointeri

- Operatorul de dereferentiere (indirectare) * : **`int *p;`**
  - **`p`** este pointer, **`*p`** este valoarea variabilei ce are adresa **`p`**
  - Valoarea directa a lui **`p`** este adresa unei locatii iar **`*p`** este valoarea indirecta a lui **`p`** : ceea ce este memorat in locatie

```
int a = 1, *p;
p = &a;
```



Pointeri

```
int i = 3, j = 5, *p = &i, *q = &j, *r;
double x;
```

Expresia	Echivalent	Valoare
`p == &i`	`p == (&i)`	1
`**&p`	`*(*(&p))`	3
`r = &x`	`r = (&x)`	eroare!
`3**p/*q+2`	`(((3*(*p)))/(*q))+2`	3
`*(r=&j)*=*p`	`(*(r=(&j)))*=(*p)`	15

Pointeri

```
int *p; float *q; void *v;
```

Expresii corecte	Expresii incorecte
`p = 0;`	`p = 1;`
`p = (int*)1 ;`	`v = 1;`
`p = v = q;`	`p = q;`
`p = (int*)q;`	`&3;`
`q = (float*)v;`	`&(k+8);`
`*((int*)333);`	`*333;`

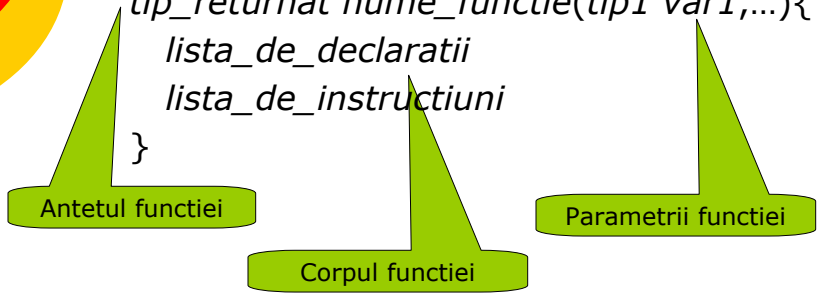
Pointeri

```
#include <stdio.h>
int main(void){
    int i=5, *p = &i;
    float *q;
    void *v;
    q = (float*)p;
    v = q;
    printf("p = %p, *p = %d\n", p, *p);
    printf("q = %p, *q = %f\n", q, *q);
    printf("v = %p, *v = %f\n", v, *((float*)v));
    printf("(int*)456 = %p, *((int*)456) = %d\n", (int*)456,
                                        *((int*)456));
    return 0;
}
/*
p = 8fb3c, *p = 5
q = 8fb3c, *q = 0.000000
v = 8fb3c, *v = 0.000000
(int*)456 = 1c8, *((int*)456) = 0
*/
```

Functii

- Definitia unei functii:

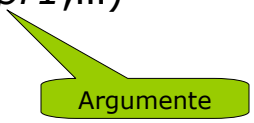
tip_returnat nume_functie(*tip1 var1*,…){
  lista_de_declaratii
  lista_de_instructiuni
}



Functii

- Apelul unei functii:

nume_functie(*expr1*,…)



- Argumentele sunt expresii ce substitue parametrii la un apel: parametrii functiei sunt initializati cu valorile argumentelor.

Functii

```
#include <stdio.h>
void swap(int x, int y){
   int temp = x; x = y; y = temp;
   printf("x = %d, y = %d\n", x, y);
}

int main(void){
    int a = 2, b = 3;
    swap(a, b); // x = 3, y = 2
    printf("a = %d, b = %d\n",a, b);
    // a = 2, b = 3
    return 0;
}
```


Functii

- Transmitere prin valoare: expresiile argument sunt evaluate si valorile sunt transmise functiei; aceste valori initializeaza parametrii.
- O modalitate de a modifica valorile unei variabile de catre o functie este de a furniza acesteia adresa variabilei:

```
scanf("%d %f", &n, &x);
```

- "call-by-value" **vs** "call-by reference"
- Limbajul C poate produce efectul mecanismului "call-by reference".

Functii

```
#include <stdio.h>
void swap(int *x, int *y){
   int temp = *x; *x = *y; *y = temp;
   printf("*x = %d, *y = %d\n", *x, *y);
}

int main(void){
     int a = 2, b = 3;
     swap(&a, &b); // *x = 3, *y = 2
     printf("a = %d, b = %d\n",a, b);
     // a = 3, b = 2
     return 0;
}
```


Functii - Declarare

- Definitia unei functii tine loc de declarare daca aceasta apare inaintea oricarui apel.
- Daca un apel apare inainte definitiei functiei, aceasta trebuie declarata.
- Declararea unei functii (prototip)se face astfel:

tip_returnat nume_functie(*tip1*,*tip2*,…, *tipn*);
tip_returnat nume_functie(*tip1 var1*,*tip2 var2*,…);

```
void schimba(int*, int*);
void ordonare2(int*, int*);
void ordonare3(int*, int*, int*);
```

Functii – Definitie si apel

```
void  schimba(int *x, int *y) {
  int temp;
  temp = *x; *x = *y; *y = temp;
}
void ordonare2(int *x, int *y) {
  if (*x > *y) schimba(x, y);
}
void ordonare3(int* x, int* y, int* z){
   ordonare2(x, y);
   ordonare2(x, z);
   ordonare2(y, z);
}
int a, b, c;
ordonare3(&a, &b, &c);
```


Functii cu numar variabil de argumente

- Este posibila utilizarea functiilor ce au numar variabil de argumente:

```
suma(3, 6, 9, 4);
suma(5, 8, 5, 6, 3, 9);
max(2, 44, 22);
max(5, a, b, c, d, e);
scanf("%d %c %f", &n, &a, &x);
scanf("%d", &m);
```

- Declaratie:
  - primele argumente(cel putin unul) sunt fixe.
  - Celelate argumente sunt declarate prin mecanismul "elipsa":

```
int suma(int nr_arg, …);
int max( int nr_arg, …);
int scanf(const char *_format, ...);
```

Functii cu numar variabil de argumente

- In definitia functiei sunt utilizate patru constructii, definite in fisierul **<stdarg.h>** :
  - **va_list** declara un pointer pentru parcurgerea listei de argumente.
  - **va_start** initializeaza parcurgerea la primul argument.
  - **va_arg** ofera argumentul urmator.
  - **va_end** termina parcurgerea listei

Exemplul 1

```
#include <stdarg.h>
#include <stdio.h>
int suma(int nargs,...){
    va_list args; //declaratie
    int i, total = 0;
    va_start(args, nargs); //initializare
    for (i = 0; i < nargs; i++)
        total += va_arg(args, int); //val.arg.+
  inaintare
    va_end(args); // eliberare
    return total;
}
int main(){
    printf("%d\n", suma(7, 1,2,3,4,5,6,7)); //=28
    printf("%d\n", suma(3, 1,2,3)); //=6
    return 0;
}
```

Exemplul 1 - Explicatii

```
va_list args;
```

- Declaratie: args este declarata variabila pointer (de tip **void***)

```
va_start(args, nargs);
```

- Initializare: variabila **args** se initializeaza cu adresa argumentului urmator lui **nargs: args = &nargs + 1**

```
va_arg(args, int);
```

- Functie ce returneaza valoarea argumentului curent si modifica pointerul **args**; acesta va pointa la urmatorul argument

```
va_end(args}
```

- se elibereaza memoria alocata pentru pointerul **args**

Exemplul 2

```
#include <stdio.h>
#include <limits.h>
#include <stdarg.h>
int max(int nargs,...){
    va_list args;
    int i, maxloc = INT_MIN, x;
    va_start(args, nargs);
    for (i = 0; i < nargs; i++)
        if ((x = va_arg(args, int)) > maxloc)
           maxloc = x;
    va_end(args);
    return maxloc;
}
int main(){
    printf("%d\n", max(7, 1,2,3,4,5,6,7));
    printf("%d\n", max(3, 1,2,3));
    printf("%d\n", max(0));
    return 0;
}
```

Curs 6

- Domeniul de vizibilitate (scope)
- Clase de alocare a memoriei
- Programe mari. Module
- Functii recursive

Domeniul de vizibilitate – “scope”

- Un nume (variabila, functie) poate fi utilizat numai dupa ce a fost declarat. Declaratia descrie proprietatile numelui
- Domeniul de vizibilitate (*scope*) al unui nume este multimea instructiunilor (liniilor de cod) in care poate fi utilizat acel nume
- Regula de baza: identificatorii sunt accesibili doar in blocul in care au fost declarati; ei sunt necunoscuti in afara acestor blocuri.
- variabile globale – variabile ce sunt declarate in afara oricarui bloc
- variabile locale sunt cele declarate:
  - in functii
  - in blocuri
  - ca parametri

Domeniul de vizibilitate – “scope”

- Blocuri:
  - paralele: {…}…{…}. In acest caz cel de-al doilea bloc “nu stie” nimic de variabilele declarate in primul bloc. Functiile sunt declarate “in paralel”
  - cuibarite: {…{…}…}. Un nume declarat in blocul exterior este vizibil in cel interior daca nu este redefinit aici; in acest caz numele din blocul exterior este “ascuns” sau “mascat”. Spunem ca fiecare bloc are “propria nomenclatura” pentru numele variabilelor

Exemplul 1

```
{
   int a, b;
   …
   {
        float b;
        … /* int a cunoscut, int b necunoscut */
   }
   …
   {
        float a;
        …       /* int b cunoscut dar int a nu */
   }
   …
}
```

Exemplul 2

```
{
   int a = 1, b = 2, c = 3;
   printf("%2d%2d%2d\n", a, b, c);          /* 1 2 3 */
   {
        int b = 4; float c = 5.0f;
        printf("%2d%2d%4.1f\n", a, b, c); /* 1 4 5.0 */
        a = b;
        {
           int c; c = b;
           printf("%2d%2d%2d\n", a, b, c);  /* 4 4 4 */
        }
        printf("%2d%2d%4.1f\n", a, b, c); /* 4 4 5.0 */
   }
   printf("%2d%2d%2d\n", a, b, c);         /* 4 2 3 */
}
```


Clase de alocare

- Zona de memorie utilizata de un program C cuprinde 4 subzone:
  - Zona text: codul programului
  - Zona de date: variabilele globale
  - Zona stiva: date temporare (variabilele locale)
  - Zona heap: memoria dinamica
- Clasele de alocare a variabilelor:
  - Statica: variabile implementate in zona de date
  - Auto: variabile implementate in stiva
  - Dinamica: variabile implementate in heap , alocate dinamic
  - Register: variabile implementate intr-un registru de memorie

Alocarea implicita

	Variabile globale	Variabile locale
Alocare	statica la compilare	auto la executie bloc
Durata de viata	cea a intregului programul	cea a blocului in care e declarata
Initializare	cu zero	nu

Clase de alocare

- Se poate utiliza cuvantul cheie auto in declararea variabileor locale:

```
auto int a, b, c;
auto double f;
```

- Clasa de alocare extern:o variabila (globala) sau o functie declarata extern este vizibila si in alt fisier decat cel in care a fost declarata
- Functiile au clasa de alocare extern; cuvantul cheie extern poate fi utilizat la declararea/definirea functiilor:

```
extern double sinus(double);
```

Clase de alocare - exemplu

```
/* fisierul main.c */
int a = 1, b = 2, c = 3; /* variabile globale */
int f(void);  /* prototip */
int main(void){
  printf("a = %d, b = %d, c = %d, f() = %d\n",
                                a, b, c, f());
}

/* fisierul f.c */
int f(void){
  extern int a; /* cauta a in afara fisierului */
  int b, c; /* b, c locale */
  a = b = c = 22;
  return (a + b + c);
}
```


Clasa de alocare static - local

- O variabila locala declarata static are durata de viata egala cu cea a programului: la intrarea in bloc valoarea sa este cea care a avut-o la iesire:

```
int f(void){
 static int contor = 0;
 return contor++;
}
f(); f(); f();
```

Exemplu

```
#include <stdio.h>
int f(void);
int main(void) {
  int i;
  for (i=0; i<10; i++){
     if(!(i%3))
          printf("\nFunctia f() este apelata a %d-a oara.", f());
  }
  return 0;
}
int f(void) {
  static int nr_apeluri=0;
  nr_apeluri++;
  return nr_apeluri;
}
/*
Functia f() este apelata a 1-a oara.
Functia f() este apelata a 2-a oara.
Functia f() este apelata a 3-a oara.
Functia f() este apelata a 4-a oara.
*/
```


Clasa de alocare static - extern

- O variabila globala declarata static are domeniul de vizibilitate redus la fisierul sursa in care este declarata, doar dupa declaratia sa:

```
int f(void){
 /*variabila v nu este vizibila*/
}
static int v;
void g(void){
 /* v este vizibila aici */
}
```

- O functie definita/declarata static este vizibila doar in fisierul in care apare definitia sa.

Clasa de alocare static - extern

```
static int g(void); /* prototip */
void f(int a){
  …
  /* g este vizibila aici */
}
static int g(void){
…
}
/* g nu este vizibila in alt
fisier */
```


Clasa de alocare register

- O variabila declarata register solicita sistemului alocarea ei intr-un registru masina, daca este posibil
- Se utilizeaza pentru variabile foarte solicitate, pentru marirea vitezei de executie:

```
{
    register int i;
    for(i = 0; i < N; ++i){
         /*… */
    }
} /* se elibereaza registrul */
```

Domeniul de vizibilitate - rezumat

- Intr-un fisier (resp. bloc) un identificator este vizibil dupa declararea sa pana la sfarsitul fisierului (resp. blocului) cu exceptia blocurilor(resp. subblocurilor) in care este redeclarat
- Definitia unui identificator mascheaza pe cea a aceluiasi identificator declarat intr-un suprabloc sau in fisier (global)
- Aparitia unui identificator face referinta la declararea sa in cel mai mic bloc (sau fisier) care contine aceasta aparitie
- Functiile si variabilele globale ale unei unitati de program (fisier) sunt implicit *publice*: sunt accesibile din alte unitati de program.
- **`extern`** indica o declaratie fara definire: permite referirea unei variabile globale definita in afara unitatii de program.
- **`static`** face ca o variabila globala sau o functie sa fie *privata(proprie)* unitatii unde a fost definita: ea devine inaccesibila altei unitati, chiar prin folosirea lui **`extern`**.

Dezvoltarea de programe mari; Module

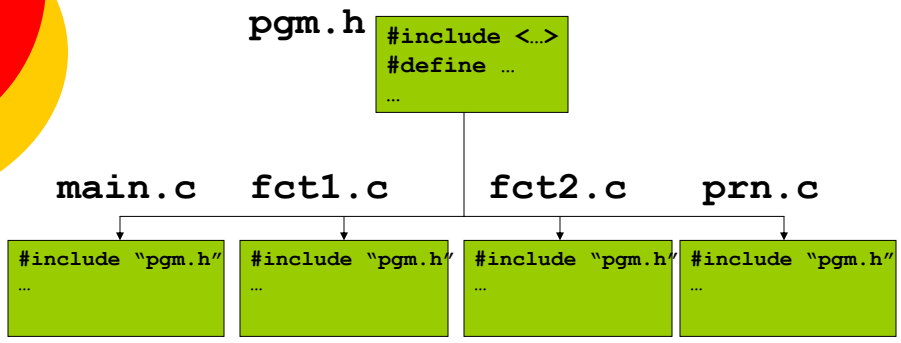
- Un program "mare" este scris ca o colectie de :
  - Fisiere .h ce contin declaratiile de variabile globale si declaratiile de functii
  - Fisiere .c care contin definitii de functii
  - Fisierul main.c care contine apelul functiilor construite in scopul rezolvarii problemei

Dezvoltarea de programe mari; Module

Metodologia recomandata:

- Se scrie un fisier header *fisier*.h care sa contina directive **`#include`**, **`#define`**, prototipurile functiilor folosite in aplicatie, declaratii de variabile globale etc. Acest fisier va fi inclus in fiecare unitate de program (fisier .c)
- Fisierele .c reprezinta implementarea functiilor declarate in fisierul header si pot contine variabile si functii private (declarate static), inaccesibile altor unitati de program

Module



Dezvoltarea de programe mari; Module

```
/* Fisierul pgm.h */
#include <stdio.h>
#include <stdlib.h>

#define   N   3

void   fct1(int k);
void   fct2(void);
void   prn_info(char *);
```


Dezvoltarea de programe mari; Module

```
/* Fisierul main.c */
#include "pgm.h"
int main(void){
   char   ans;
   int    i, n = N;
   printf("%s",
      "Exemplu de program modular.\n"
      "Doriti informatii?  ");
   scanf(" %c", &ans);
   if (ans == 'y' || ans == 'Y')
      prn_info("progmare");
   for (i = 0; i < n; ++i)
      fct1(i);
   printf("Bye!\n");
   return 0;
}
```

Dezvoltarea de programe mari; Module

```
/* Fisierul fct1.c */
#include "pgm.h"
void fct1(int n){
   int   i;
   printf("Hello from fct1()\n");
   for (i = 0; i < n; ++i)
      fct2();
}

/* Fisierul fct2.c */
#include "pgm.h"
void fct2(void){
   printf("Hello from fct2()\n");
}
```


Dezvoltarea de programe mari; Module

```
/* Fisierul prn.c */

#include "pgm.h"
void prn_info(char *pgm_name){
   printf("Folositi:  %s\n\n", pgm_name);
   printf("%s\n","Programul ilustreaza modul"
   "in care se poate scrie un program in\n"
   "mai multe fisiere. In acest exemplu avem "
   "un fisier header care este inclus in cele\n"
   "trei fisiere .c: fct1.c,fct2.c si main.c\n");
}
```

Recursie. Functii recursive

- Functia f() apeleaza direct functia g() daca in definitia lui f() exista un apel la g()
- Functia f() apeleaza indirect functia g() daca f() apeleaza direct o functie h() iar h() apeleaza direct sau indirect functia g()
- Functia f() este definita recursiv daca ea se autoapeleaza direct sau indirect

Recursie. Functii recursive

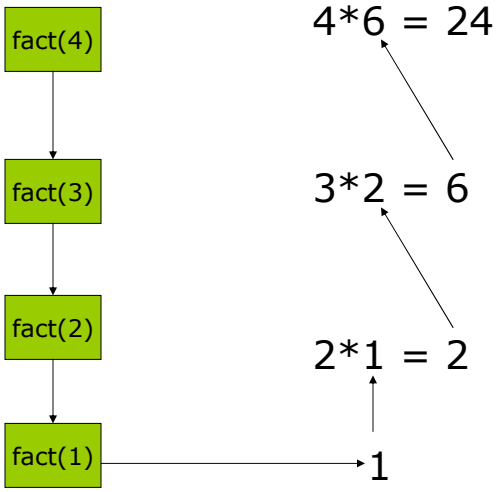
- Definitia unei functii recursive cuprinde:
  - Testarea cazului de baza – conditia de oprire a apelului recursiv
  - Apelul recursiv (cazul general): o variabila (intreaga) este transmisa ca parametru functiei insasi, in asa fel ca dupa un numar de pasi sa se atinga cazul de baza
  - Exista si functii recursive fara parametri

Functii recursive - Exemplu

- Definitia functiei factorial:
  - Cazul de baza: 0! = 1
  - Cazul general: n! = n*((n-1)!), n>0

```
int factorial(int n){
  if(n <= 1)
        return 1;
  else
        return (n*factorial(n-1));
}
```


Apelul recursiv:



Exemplu de apel fara sfarsit

```
#include <stdio.h>
int main(void){
  printf("Universul este infinit!");
  main();
  return 0;
}
```


factorial(n) - varianta iterativa

```
int factorial(int n){
  int produs = 1;
  for(; n > 1; --n)
     produs *= n;
  return produs;
}
```

- Valoarea returnata de factorial(n) este corecta doar pentru valorile lui n pentru care n! <= INT_MAX

Functie recursiva fara parametri

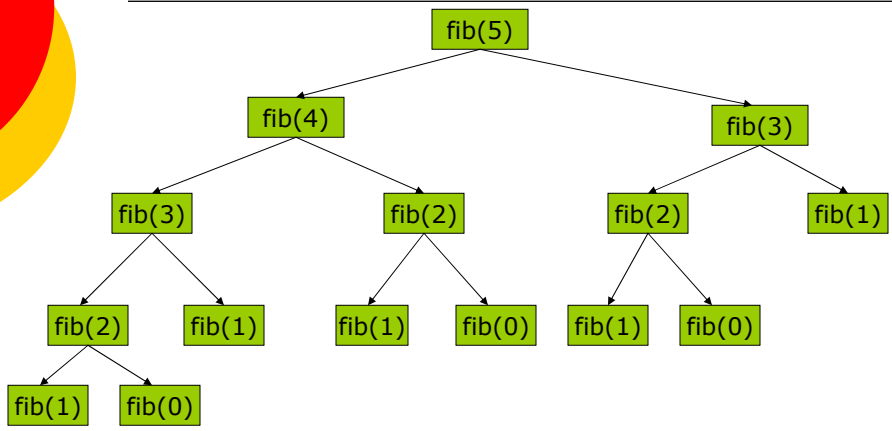
```
#include <stdio.h>
void scrie(void);
void main(void){
   printf("Introdu un text pe o linie:\n");
   scrie();  /*  scrierea textului in ordine inversa */
   getchar();
}
void scrie(void){                /* functie recursiva  */
   int c;
   if((c = getchar()) != '\n')  /* conditia de oprire */
        scrie();                 /* apel recursiv */
   putchar(c);
}
/*
Introdu un text pe o linie:
Textul acesta este un test pentru recursie

eisrucer urtnep tset nu etse atseca lutxeT
*/
```


Recursie vs. iteratie: Fibonacci recursiv

- $f_0 = 0$, $f_1 = 1$,
- $f_n = f_{n-1} + f_{n-2}$, $n > 1$

```
long int fib(int n){
  if(n <= 1)
     return (long)n;
  else
     return(fib(n-1) + fib(n-2));
}
```

Fibonacci recursiv: arbore apeluri



Numar de apeluri

n	fib(n)	apeluri
2	1	3
24	46368	150049
42	267 914 296	866 988 873
43	433 494 437	1402 817 465

Recursie vs. iteratie: Fibonacci iterativ

```
long int ifib(int n){
  int k;
  long   f0 = 0, f1 = 1, temp;
  if(n <= 1) return (long)n;
  else
  for (k = 2; k <= n; ++k) {
      temp = f1;
      f1 += f0;
      f0 = temp;
  }
  return f1;
}
```


Turnurile din Hanoi

```
void muta(int n, char a, char b, char c){
  if (n == 1)
  printf("%c -> %c; ",a, b);
  else {
     muta(n-1, a, c, b);
     printf("%c -> %c; ",a, b);
     muta(n-1, c, b, a);
  }
}
```

Cel mai mare divizor comun

```
int cmmdc(int m, int n){
   if(m == n) return m;
   else{
      if(m > n)return cmmdc(m-n, n);
      else return cmmdc(m, n-m);
   }
}
```

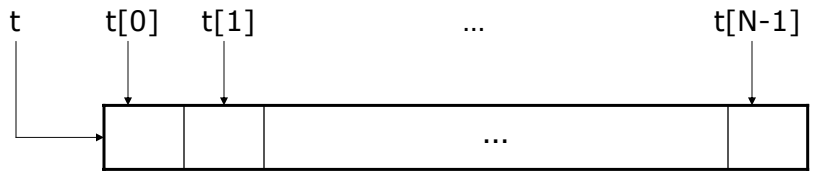
Curs 7

- Tablouri si pointeri
  - Tablouri cu o singura dimensiune
    - Declarare
    - Aritmetica pointerilor
    - Tablourile ca parametri
    - Initializarea tablourilor
  - Tablouri cu doua dimensiuni
  - Alocare dinamica
  - Siruri

Tablouri cu o singura dimensiune

- Declaraera tablourilor

```
tip nume_tablou[marime];

#define N 7
int t[N];
```



Tablouri cu o singura dimensiune

- Numele tabloului este un nume de variabila
- Numele tabloului este pointer constant catre primul element din tablou:

```
t     echivalent cu    &t[0]
t+i   echivalent cu    &t[i]
t[i] echivalent cu  *(t+i)
*t     echivalent cu t[0]
*(t+i) echivalent cu t[i]
```


Tablouri cu o singura dimensiune

- Parcurgerea tablourilor:

```
#define n 20
int t[n], *p;
suma = 0;
```

```
for(i=0; i<n; i++)
   suma +=  t[i];
```

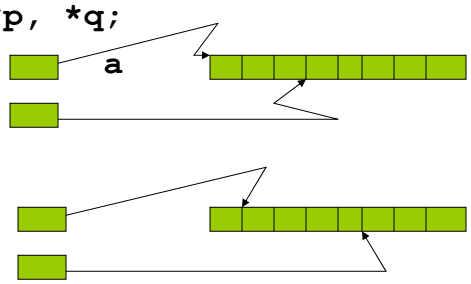
```
for(i=0; i<n; i++)
    suma += *(t+i);
```

```
p = t;
for(i=0; i<n; i++)
    suma +=  p[i];
```

```
for(p=t; p<&t[n]; p++)
    suma += *p;
```

Aritmetica pointerilor

```
int i=4, a[8], *p, *q;
p = a;
q = p + 3;
p+i;
++p;
p += i;
p - q;        = nr. elemente
(int)p - (int)q;
 = nr.elem*sizeof(int)= nr.octeti
```



Initializarea tablourilor

```
int a[5] = {1, 2, 3, 4, 5};
float f[] = {0.0,1.0,2.0,3.0,4.0};
int a[100] = {0};
int b[7] = {1, 2, 3};
char s[] = "info";
char s[] = {'i','n','f','o','\0'};
```

- Tablourile globale sau static sunt inițializate cu zero dacă nu sunt inițializate explicit.

Acces la elemente prin indici

- Accesul la elementele tabloului se face prin expresiile de indice: nu se face control asupra depasirii valorilor expresiilor de indice
- Nu exista operatii globale asupra tablourilor: nu se poate testa direct, cu == egalitatea a doua tablouri, nu se pot copia cu =

Exemplu

```
int a[N], *pi, *qi;  float b[N], *pf, *qf;  double c[N], *pd, *qd;
char  d[N], *pl, *ql;

printf("Pointeri int:\n");
pi = a; qi = pi+1;
printf("  qi-pi=%d\n",qi-pi);
printf("  sizeof(int) = %d, (int)qi-(int)pi = %d\n",sizeof(int),
  (int) qi - (int) pi);
printf("Pointeri float:\n");
pf = b;qf = pf+1;
printf("  qf-pf=%d\n",qf-pf);
printf("  sizeof(float) = %d, (int)qf-(int)pf = %d\n",sizeof(float),
  (int) qf - (int) pf);
printf("Pointeri double:\n");
pd = c;qd = pd+1;
printf("  qd-pd=%d\n",qd-pd);
printf("  sizeof(double) = %d, (int)qd-(int)pd =
  %d\n",sizeof(double), (int) qd - (int)pd);
printf("Pointeri char:\n");
pl = d;ql = pl+1;
printf("  ql-pl=%d\n",ql-pl);
printf("  sizeof(char) = %d, (int)ql-(int)pl = %d\n", sizeof(char),
  (int) ql - (int) pl);
```

Exemplu

```
 /*
Pointeri int:
  qi-pi=1
  sizeof(int) = 4, (int)qi-(int)pi = 4
Pointeri float:
  qf-pf=1
  sizeof(float) = 4, (int)qf-(int)pf = 4
Pointeri double:
  qd-pd=1
  sizeof(double) = 8, (int)qd-(int)pd = 8
Pointeri char:
  ql-pl=1
  sizeof(char) = 1, (int)ql-(int)pl = 1
*/
```


Tablourile ca parametri

- Un tablou ce apare ca parametru formal intr-o functie este de fapt pointer:

  ```
  double suma(double a[], int n);
  ```

  este echivalent cu:

  ```
  double suma(double *a, int n);
  ```

- Un tablou – argument in invocarea (apelul) unei functii este transmis "prin adresa": in momentul apelului este transmisa adresa primului element al tabloului(transmitere prin valoare a adresei):
  - Nu se face o copie a elementelor tabloului la apel
  - Functia actioneaza direct pe tabloul argument

```
double suma(double a[], int n);
double suma(double *a, int n);

suma(v, 100);
suma(v, 8);
suma(&v[4], k-6);
suma(v+4, k-6);
```


Tablouri cu mai multe dimensiuni

- Un tablou cu n dimensiuni este un tablou (cu o dimensiune) in care tipul de baza este un tablou cu n-1 dimensiuni
- O matrice (int) **mat** de dimensiune 2x4:

5	3	- 2	1
0	12	7	4

- In C aceasta este un vector care are elemente vectori:

mat[0]	5	3	- 2	1
mat[1]	0	12	7	4

Declarare, Initializare

- Declarare:

```
int mat[2][4];
```

- Declarare cu initializare:

```
int mat[2][4]={{5, 3, -2, 1}
               {0, 12, 7, 4}};
int mat[2][4]={5, 3, -2, 1
               0, 12, 7, 4};
double a[4][5] = {0.0}
```

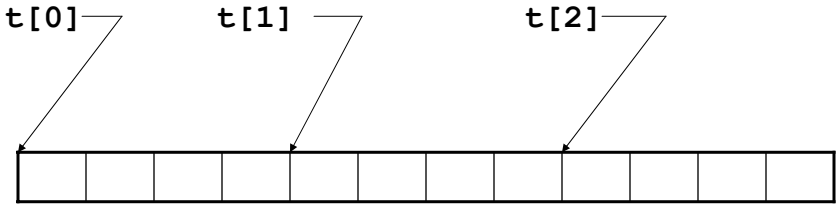

Declarare, Initializare

```
int a[2][3] = {1, 2};
int a[2][3] = {{1, 2, 3}, {0}};
int b[2][2][3] = {
          {{1, 2, 0}, {4, 0, 0}},
          {{3, 0, 0}, {0, 5, 1}}
};
int b[2][2][3] = {
          {{1, 2}, {4}},
          {{3}, {0, 5, 1}}
};
```

Zona de memorie

```
int t[3][4];
```

`t` este adresa (pointer constant) de inceput a tabloului



Accesul la elemente prin indici

```
t[1][3]   echivalent cu   *(t[1] +3)

&t[1][3]  echivalent cu   t[1] + 3

t[i][j]   echivalent cu:
          *(t[i] + j)
          (*(t+i))[j]
          *(*(t+i)+j)
          *(&t[0][0]+4*i+j)
```

Functia de alocare a memoriei

```
int a[M][N];
```

`a[i][j]` **echivalent cu:**

```
(&a[0][0] + N*i + j)
```

```
int a[M][N][P];
```

`a[i][j][k]` **echivalent cu:**

```
(&a[0][0][0] + N*P*i + P*j + k)
```


Parcurgerea tablourilor

```
int a[m][n], suma = 0;
for(i = 0; i < m; i++)
  for(j = 0; j < n; j++){
     scanf("%d", &t[i][j];
     suma += t[i][j];
/* suma += (*t[i]+j);     */
/* suma += *(*(t+i)+j);   */
/* suma += (*(t+i))[j];   */
  }
```

Tablourile cu mai multe dimensiuni ca parametri

- Trebuiesc specificate toate mărimile dimensiunilor, cu excepţia primei dimensiuni

```
int suma(int a[][NMAX], m);
int suma(int (*a)[NMAX], m);
```

- La apel se transmite (prin valoare) adresa de inceput a tabloului:

```
suma(a, m);
```


Exemplu

```
#define m 5
#define n 6
#define p 4
int suma2(int[][n], int);
int suma3(int[][n][p], int);

int main(void){
   /* Declaratii */
   printf("Tabloul b: \n");
   for(i=0; i < m; ++i){
        for(j = 0; j < n; ++j)
               printf("%d  ", b[i][j]);
        printf("\n");
   }
   printf("Suma elementelor sale: %d\n", suma2(b, m));
   printf("Tabloul c: \n");
   for(i=0; i < m*n*p; ++i)
        printf("%d  ", *(c[0][0]+i));
   printf("\nSuma elementelor sale: %d\n", suma3(c, m));
}
```

Exemplu (cont)

```
int suma2(int a[][n], int x){
   int   i, j, suma = 0;
   for (i = 0; i < x; ++i)
      for (j = 0; j < n; ++j)
         suma += a[i][j];
   return suma;
}
int suma3(int a[][n][p], int x){
   int   i, j, k, suma = 0;
   for (i = 0; i < x; ++i)
      for (j = 0; j < n; ++j)
         for (k = 0; k < p; ++k)
       suma += a[i][j][k];
   return suma;
}
```


Alocare dinamică a memoriei

`void* calloc(size_t n, size_t` *el_size*`);`

- Returneaza un pointer la o zona pentru un tablou cu n elemente fiecare ocupand *el_size* octeti
- Returneaza 0 (NULL) in caz de esec
- Initializeaza zona de memorie cu 0

```
p = calloc(n, sizeof(tip));
```

`void* malloc(size_t` ***size***`);`

- Returneaza un pointer la o zona de *el_size* octeti
- Returneaza 0 (NULL) in caz de esec
- Zona de memorie nu este initializata

```
p = malloc(n*sizeof(tip));
```

Alocare dinamică a memoriei

`void* free(void*` *var_ptr*`);`

- Elibereaza zona de memorie de la adresa *var_ptr* (alocata cu calloc() sau malloc())
- Daca *var_ptr* = NULL, nu are efect

`void* realloc(void* var_ptr, size_t size);`

- Modifica marimea obiectului pointat de **`var_ptr`** si returneaza pointer la o zona de **`size`** octeti
- Returneaza NULL in caz de esec
- Octetii alocati inainte sunt copiati

Alocare dinamică a memoriei

```
double *pd;
pd = (double*)calloc(1, sizeof(double));
                    sau
pd = (double*)malloc(sizeof(double));

int *pi, n = 10;
pi = (int*)malloc(n*sizeof(int));

void* aloc_mem(unsigned m, unsigned size){
  return malloc(m*size))
}
char *pc, Lmax = 30;
pc = (char*)calloc(Lmax, sizeof(char));
```

Alocare tablouri 1

- Alocare

```
double **a;
a = (double **)calloc(m, sizeof(double*));
for (i=0; i<n; i++)
    a[i] = (double *)calloc(n, sizeof(double));
a[1][2] = 3.14;
printf("%lf", a[1][2]);
```

- Dealocare (eliberare)

```
for (i=0; i<n; i++)
    free(a[i]);
free(a);
```


Alocare tablouri 2

- Alocare

```
double *p;
p=(double *)calloc(15, sizeof(double));
a = (double **)calloc(3, sizeof(double*));
for (i=0; i<3; i++)
    a[i] = p+i*5;
a[1][2] = 3.1415;
```

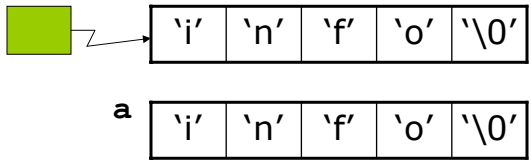
- Dealocare (eliberare)

```
p = (double *)a[0];
free(p);
free(a);
```

Siruri

- **Tablouri unidimensionale de tip `char`**

```
char *p = "info";
char s[] = "info";
   /* = {'i','n','f','o','\0'}; */
```



`p+i, ++p, p-2` `s+i  ++s  s-2`

Macrouri si functii pentru siruri

- In fisierul `ctype.h`

`issspace(c), isdigit(c), islower(c), …`

- In fisierul `string.h`

```
char *strcat(char *s1, const char *s2);
char *strcmp(const char *s1,const char*s2);

char *strcpy(char *s1, const char *s2){
  register char *p = s1;
while(*p++ = *s2++) ;
return s1;
}

size_t strlen(const char *s);
char* strchr(const char* s, int c);
char* strdup(const char* s);
```

Exemplu

```
unsigned lung(const char *s)
{
   register int   n;

   for (n = 0; *s != '\0'; ++s)
      ++n;
   return n;
}

#include <string.h>
#include <stdio.h>
void main()
{
   char* sir = "Exemplu_de_sir_care_contine_sapte_cuvinte";
   printf(" Sirul : %s \n are lungimea %d \n", sir, lung(sir));
   printf(" Sirul : %s \n are lungimea %d \n", sir,
   strlen(sir));
}
```


Exemplu

```
char* string_copy(char* s1, register const char* s2)
{
   register char *p = s1;
   while(*p++ = *s2++)
         ;
   return s1;
}
#include <string.h>
#include <stdio.h>
void main()
{
   char* sir1 = "Exemplu de sir1 ";
   char* sir2 = "Sirul2 va inlocui sirul 1";

   printf(" Sirul : %s \n are lungimea %d \n", sir1, strlen(sir1));
   printf(" Sirul : %s \n are lungimea %d \n", sir2, strlen(sir2));
   string_copy(sir1,sir2);
   printf(" Sirul : %s \n are lungimea %d \n", sir1, strlen(sir1));
   strcpy(sir1, "Ha_Ha_Ha!");
   printf(" Sirul : %s \n are lungimea %d \n", sir1, strlen(sir1));
       printf( " Restul: %s\n", sir1+10);
}
```